Maven

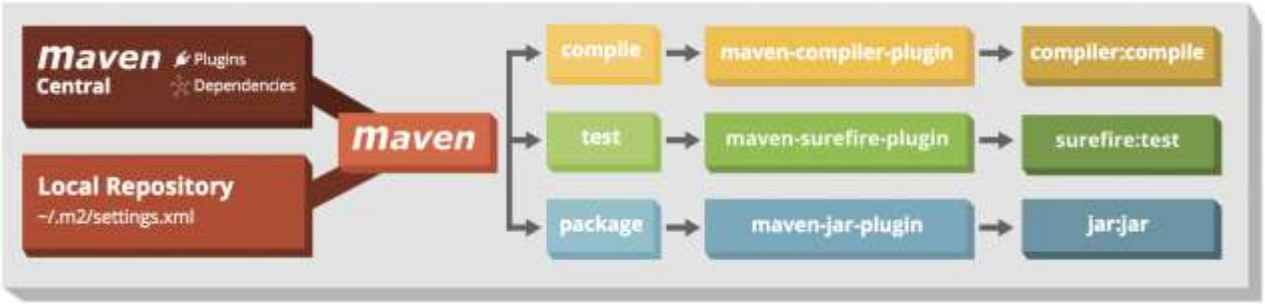
Tabla de contenidos

01

Introducción a Maven

Maven: gestión de la configuración

- Herramienta para la **gestión de proyectos** de software basada en el concepto de **POM** (Project Object Model).
- Funciones principales:
  - Compilación
  - Empaquetado
  - Generación de documentación
  - Ejecución de tests
  - Preparación de builds



Características de Maven

- Facilita la compilación y empaquetamiento
  - Herramientas que facilitan estos procesos
- Permite construir sistemas de forma uniforme
  - Configuración POM (Project Object Model) + plugins
  - Define forma estándar de estructurar código fuente, dependencias y herramientas necesarias
- Fomenta las buenas prácticas de desarrollo
  - Ejecución de tests, mantenimiento del código fuente
- Facilita la migración a nuevas características
  - Plugins declarados en el POM
- Facilita información de la calidad del proyecto
  - Integración con herramientas QA, testing, dependencias y codificación

Beneficios de utilizar Maven

- Se basa en patrones y estándares → Mejora el **mantenimiento** y la **reusabilidad.**
- Se encarga de la **gestión de dependencias** incluyendo las transitivas.
- Los desarrolladores pueden “moverse” entre proyectos ya que no necesitan aprender a compilar o empaquetar.

02

Instalación y Configuración

Instalación de Apache Maven (Linux)

1. Asegurarnos de que la variable de entorno JAVA_HOME está configurada y contiene la ruta de instalación del JDK
2. Extraer archivo de instalación en cualquier directorio:

   ```
   unzip apache-maven-3.6.3-bin.zip
   ```

   o

   ```
   tar xzvf apache-maven-3.6.3-bin.tar.gz
   ```

Instalación de Apache Maven (Windows)

1. Comprobar variable de entorno:

   ```
   echo %JAVA_HOME%
   C:\Program Files\Java\jdk1.8.0_51
   ```

2. Extraer archivo de instalación en cualquier directorio
3. Añadir ruta de instalación a la variable de entorno del sistema:

   ```
   C:\Program Files\apache-maven-3.6.3\bin
   ```

4. Verificar instalación en una consola de comandos (cmd):

   ```
   mvn --version|-v
   ```

settings.xml

Archivo de configuración principal

- Settings “global” (ubicado en carpeta de instalación de Maven)
- Settings “del usuario” (situado en ${user.home}/.m2)
- Opciones de configuración:
  - Repositorio Local (localRepository)
  - Repositorios remotos
  - Proxy
  - Usuarios y contraseñas de conexión a repositorios
  - Perfiles asociados a repositorios remotos (activeProfiles)

settings.xml

Elementos básicos

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                              https://maven.apache.org/xsd/settings-1.0.0.xsd">
          <localRepository/>
          <interactiveMode/>
          <offline/>
          <pluginGroups/>
          <servers/>
          <mirrors/>
          <proxies/>
          <profiles/>
          <activeProfiles/>
</settings>
```

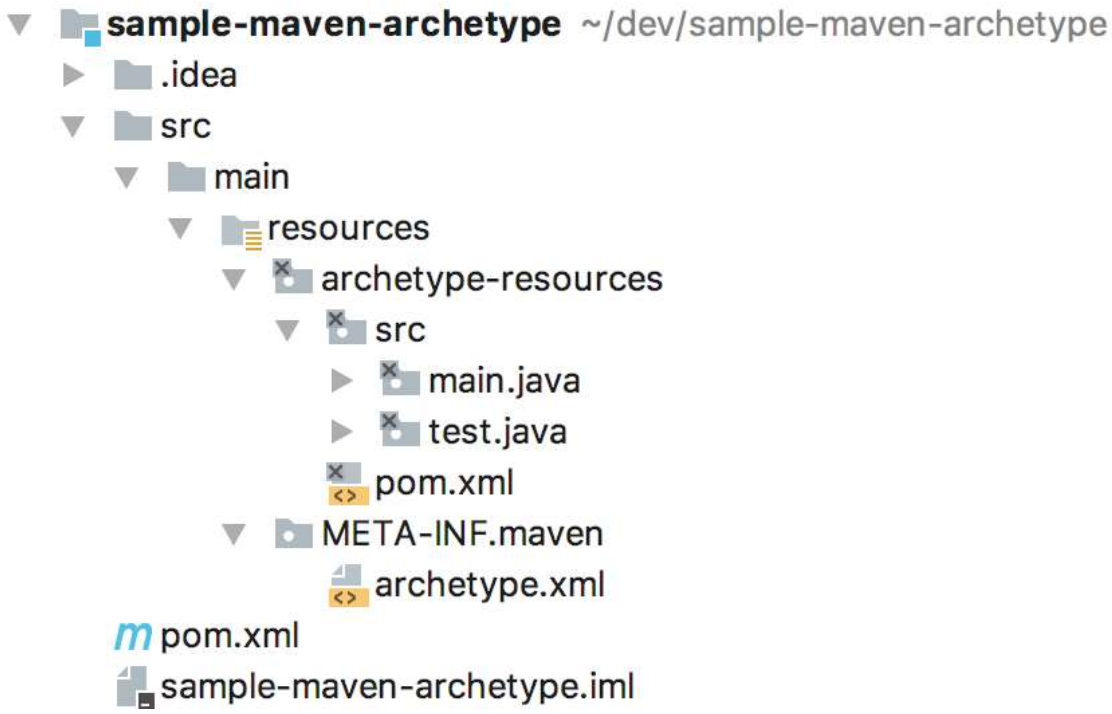
03

Arquetipos

Arquetipo

- Es una **plantilla, patrón o modelo** predefinido que ayuda a generar la estructura de paquetes básica que un tipo de aplicación debería contener.
- En desarrollo, los arquetipos permiten **reutilizar código y arquitecturas.**
- Crearemos un proyecto nuevo a partir de un **arquetipo Maven.**
- En resumen, los Arquetipos son una herramienta de plantillas para proyectos.

Arquetipo básico de Maven

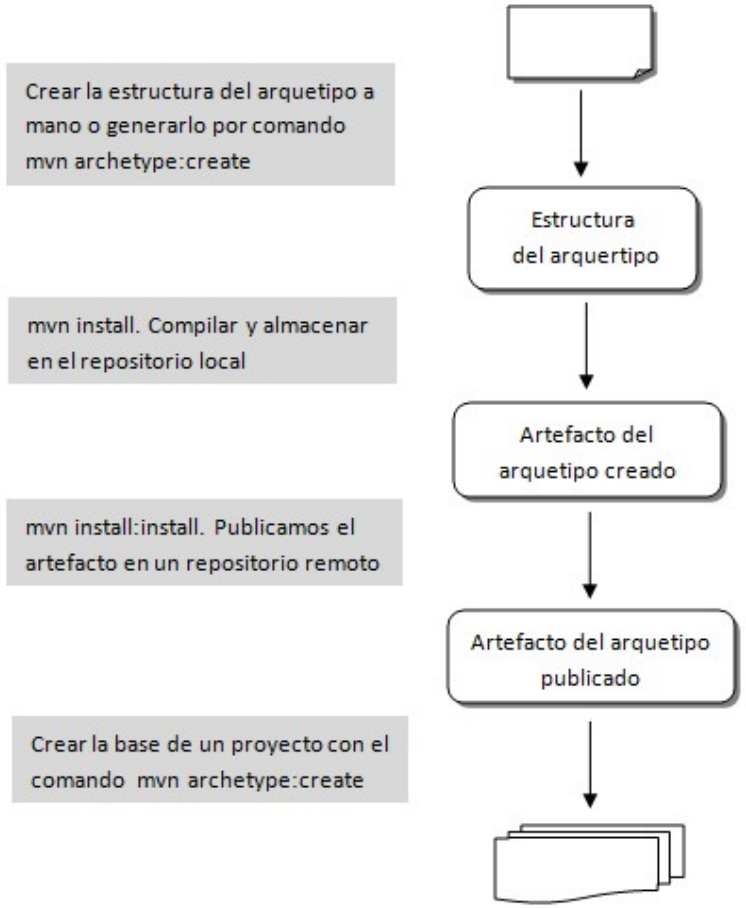


Catálogo de Arquetipos Maven

- maven-archetype-archetype
- maven-archetype-webapp
- maven-archetype-quickstart
- maven-archetype-j2ee-simple
- maven-archetype-plugin-site
- maven-archetype-simple
- maven-archetype-portlet
- …

Ciclo de vida de un Arquetipo Maven

- Pasos y comandos habituales de creación y uso de arquetipos



Beneficios de usar Arquetipos

- Consistencia entre diferentes proyectos con tecnologías similares
- Reutilización, composición y ajuste de arquetipos
- Evitan tener que configurar las opciones iniciales del proyecto (librerías, dependencias, configuraciones…)
- Estandarización, centralización y compartición de proyectos dentro de una organización
- Reduce tiempos de inicio/comienzo de la implementación (entorno previamente configurado)
- La estructura del proyecto facilita al equipo las tareas de desarrollo, distribución, portabilidad y despliegue.

04

Creación de un proyecto con Maven

Creación de un proyecto con Maven

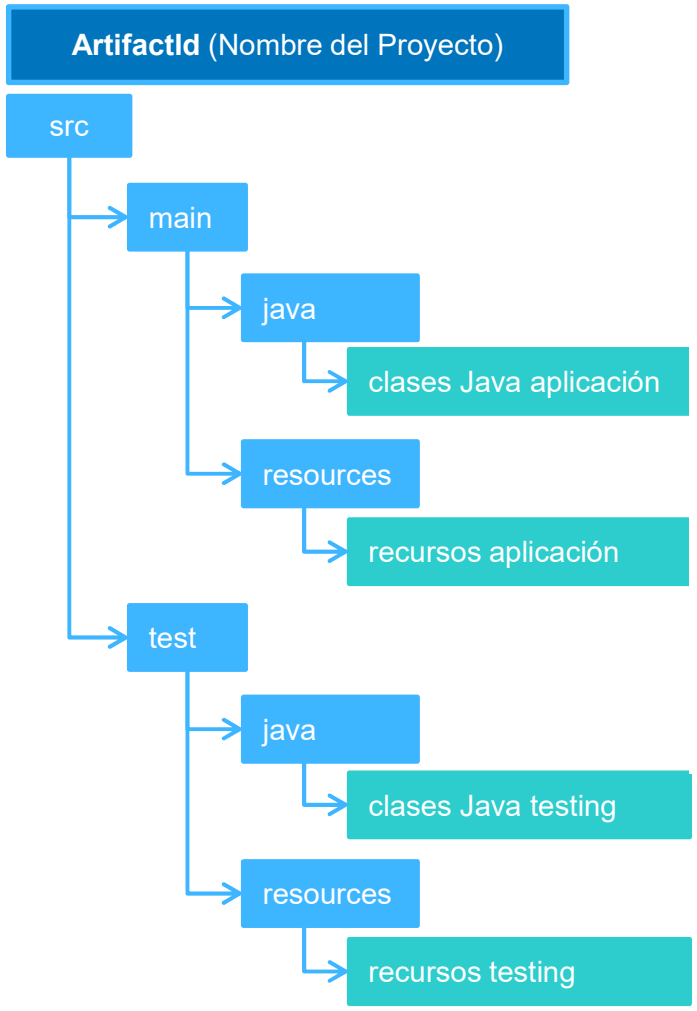
- Crear un proyecto Maven (comando mvn o IDE)
- Elegir el **arquetipo** del catálogo de arquetipos de Maven
- Configurar datos del proyecto (**archetype parameters**)
  - **Group Id:** Grupo al que pertenece el proyecto (lib, backend, component,…)
  - **Artifact Id:** Nombre del proyecto
  - **Version:** Versión del proyecto (por defecto 0.0.1-SNAPSHOT)
  - **Package:** Tipo de empaquetado (jar, war,…)

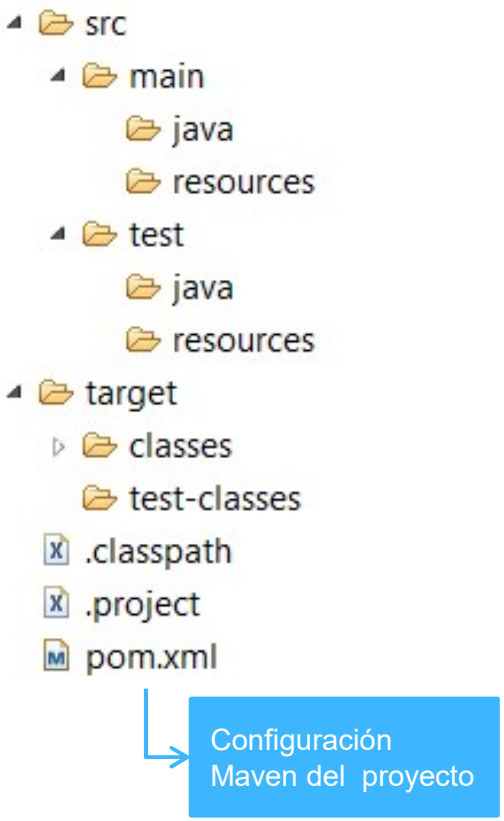
Estructura del proyecto Maven

- Maven crea una ubicación estándar para el código fuente del proyecto divido en:
  - Código Java de la aplicación
  - Recursos de la aplicación
  - Código Java para testing
  - Recursos para testing
- Esta estructura permite que no se incluya el código de testing al empaquetar el proyecto.

Estructura del proyecto maven generado

Ejemplo de proyecto simple (sin selección de arquetipo)





05

Integración con IntelliJ IDEA

Integración con IntelliJ IDEA

- IntelliJ IDEA tiene compatibilidad funcional completa con Maven
- Ayuda a la automatización de los ‘builds’
- Funcionalidades:
  - Crear un proyecto Maven nuevo
  - Abrir un proyecto Maven existente
  - Añadir un módulo Maven nuevo a un proyecto existente
  - Configurar un proyecto Maven multi-módulo
  - Acceder a las opciones de configuración de Maven
  - Instalar versión de Maven “custom”
  - Cambiar la versión del JDK de un proyecto Maven

06

Modelo de Objetos del Proyecto: POM

Project Object Model (POM)

- Representación XML de un proyecto Maven en fichero `pom.xml`
- Descripción del proyecto (nombre, versión, bibliotecas…)
- Dependencias de otros módulos y componentes externos
- Configuración de plugins utilizados en el proceso de build

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <artifactId>my-project</artifactId>
</project>
```

El archivo pom.xml Ejemplo

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="<a
href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-
instance</a>"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <a href="http://maven.apache.org/maven-
v4_0_0.xsd">http://maven.apache.org/maven-v4_0_0.xsd</a>">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.autentia.demoapp</groupId>
    <artifactId>autentiaNegocio</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>Maven Quick Start Archetype</name>
    <url><a href="http://maven.apache.org/">http://maven.apache.org</a></url>
          …
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
             …
    </dependencies>
          …
</project>
```

El archivo pom.xml

Elementos Básicos
<groupId>...</groupId>
<artifactId>...</artifactId>
<version>...</version>
<packaging>...</packaging>
<dependencies>...</dependencies>
<parent>...</parent>
<dependencyManagement>...</dependencyManagement>
<modules>...</modules>
<properties>...</properties>

El archivo pom.xml

Configuración de Builds
`<build>...</build>`
`<reporting>...</reporting>`

El archivo pom.xml

More Project Information
<name>...</name>
<description>...</description>
<url>...</url>
<inceptionYear>...</inceptionYear>
<licenses>...</licenses>
<organization>...</organization>
<developers>...</developers>
<contributors>...</contributors>

El archivo pom.xml

Configuración de Entorno
<issueManagement>...</issueManagement>
<ciManagement>...</ciManagement>
<mailingLists>...</mailingLists>
<scm>...</scm>
<prerequisites>...</prerequisites>
<repositories>...</repositories>
<pluginRepositories>...</pluginRepositories>
<distributionManagement>...</distributionManagement>
<profiles>...</profiles>

El archivo pom.xml

- Elementos mínimos

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.engineering.bbva</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0</version>
</project>
```

El archivo pom.xml: Maven Coordinates

Elementos Obligatorios*	
`<groupId>`	Generally unique amongst an organization/project. For example, all core Maven artifacts live under groupId `org.apache.maven`
`<artifactId>`	Name of the project. Along with the groupId, fully defines the artifact's living quarters within the repository.
`<version>`	Keeps changes of code versions in line.
`groupId:artifactId:version`	

* `groupId` and `version` do not need to be explicitly defined if they are inherited from a parent

El archivo pom.xml: packaging

- Especificación del empaquetado de proyecto, por defecto JAR

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  …
  <packaging>war</packaging>
  …
</project>
```

Módulo Parent

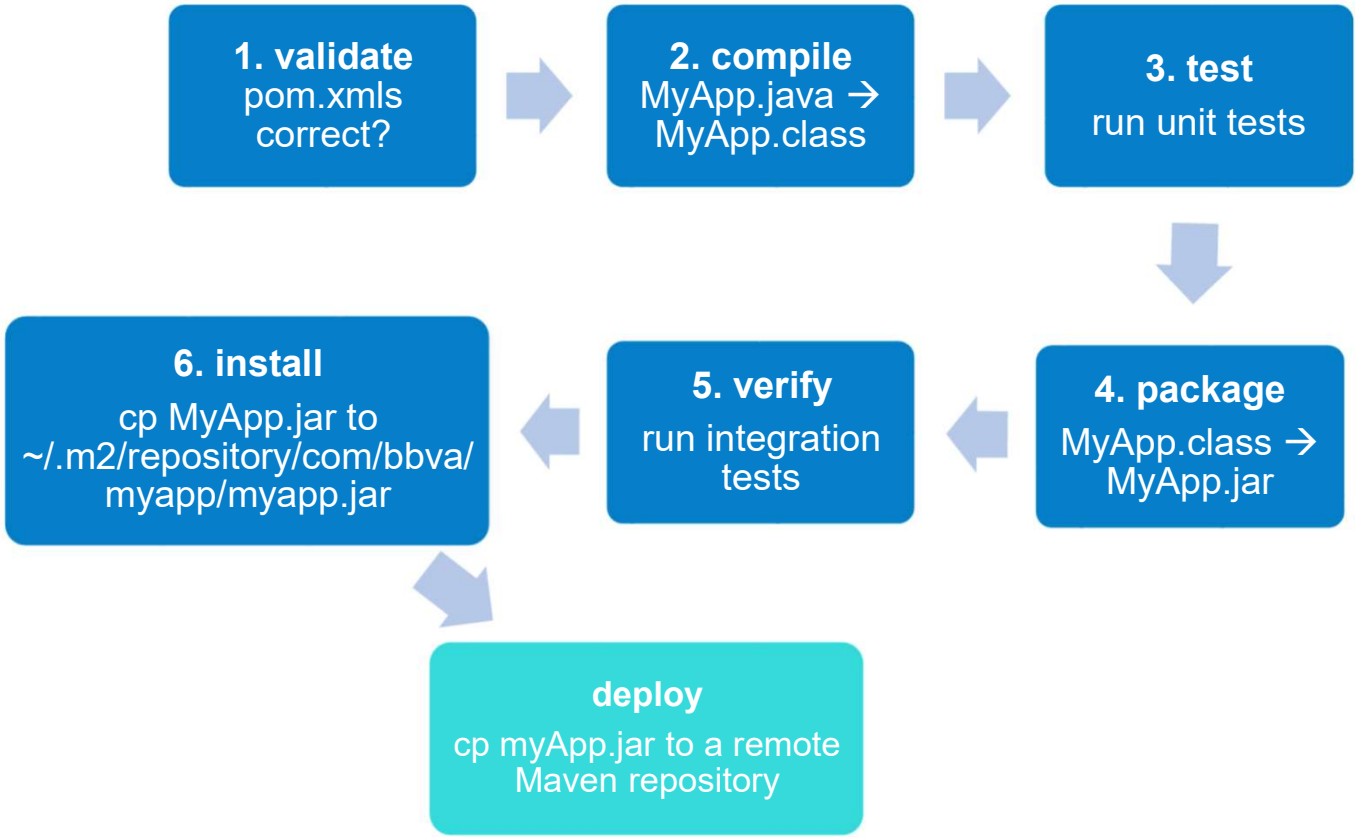
- Maven permite utilizar un POM "padre", del cual heredan los otros POM.
- En el POM padre se pueden definir elementos comunes, evitando tener que repetir configuraciones entre proyectos.

```
<parent>
  <groupId>com.demo.maven</groupId>
  <artifactId>app-parent</artifactId>
  <version>1.0.0</version>
</parent>
<artifactId>app-que-hereda</artifactId>
```

07

Ciclo de vida del proyecto

Maven's default lifecycle phases



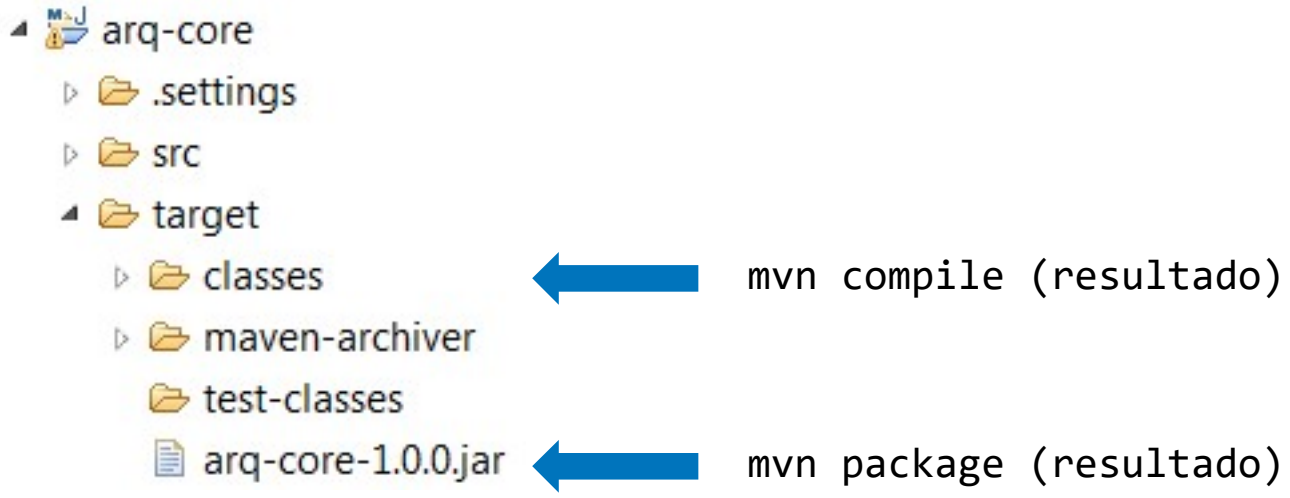
Ejecutando Apache Maven

mvn [options] [<goal(s)>] [<phase(s)>]	Syntax for running Maven
mvn -h	Ayuda y documentación
mvn **clean**	Deletes the /target folder
mvn **package**	Converts the .java source code into a .jar|.war file and puts it into the /target folder
mvn **install**	First, it does a package. Then it takes that .jar|.war file and puts it into your local Maven repository, which lives in ~/.m2/repository
mvn **verify**	Create the package and installing it in the local repository for re-use from other projects
mvn **test**	Compila los tests y los ejecuta

Compile

- mvn compile: Comando para compilar el proyecto, separando el código de test

Compile + Package



Proceso de Build (comando)

`mvn clean install`

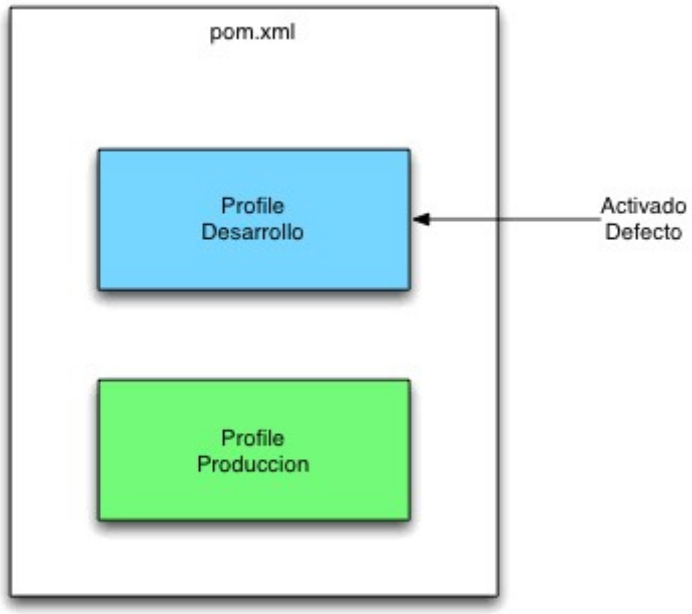
1. Borra todo el código fuente Java compilado anteriormente y los recursos (.properties, …) del proyecto.
2. Compila el proyecto
3. Ejecuta tests del proyecto
4. Empaqueta el proyecto
5. Copia el fichero de build generado .jar|.war en tu repositorio local Maven

08

Perfilado de proyectos

Maven Profiles

- Permite generar **diferentes perfiles** para el **despliegue** de las aplicaciones que van ganando en complejidad.
- Permite **configurar el flujo** de Maven dependiendo del perfil que se seleccione.



Tipos de Perfiles

Tipo	Dónde está definido
Project	pom.xml
User	Defined in the Maven-settings `%USER_HOME%/.m2/settings.xml`
Global	Defined in the global Maven-settings `${maven.home}/conf/settings.xml`
Profile descriptor	Located in project basedir `profiles.xml`

Creación de Perfiles

- Permite definir propiedades concretas para la construcción del proyecto en distintas situaciones: producción, pruebas o desarrollo

```
<project>
    ...
    <profiles>
        <profile>
            <id>build-release</id>
            <properties>
                <maven.compiler.debug>false</maven.compiler.debug>
                <maven.compiler.optimize>true</maven.compiler.optimize>
            </properties>
        </profile>
    </profiles>
    ...
</project>
```

Activación de Perfiles

- Un Profile se puede **activar** de varias formas:
  - Explícitamente
  - Basada en variables de entorno
  - Configuración del SO
  - Present or missing files
  - Mediante configuración Maven:

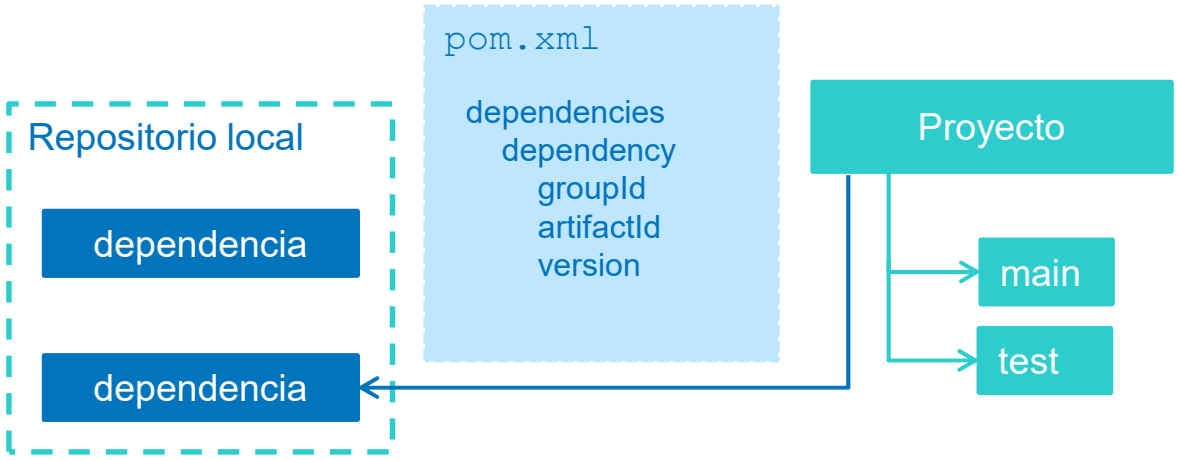
```
<settings>
...
<activeProfiles>
<activeProfile>profile-1</activeProfile>
</activeProfiles>
...
</settings>
```

09

Gestión de dependencias y repositorios

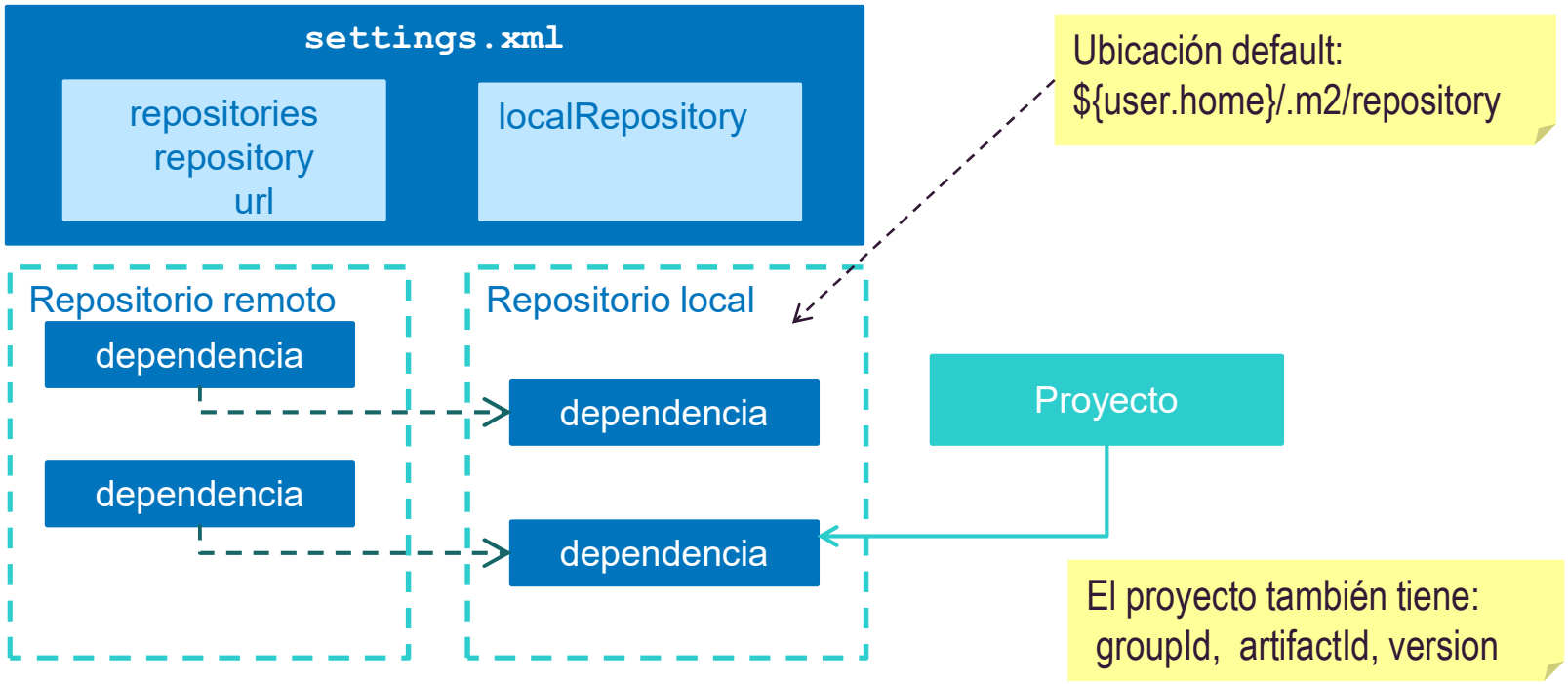
Dependencias

- Maven almacena las dependencias en el **repositorio local**



Dependencias

- Si una dependencia no está en el repositorio local, Maven intenta buscar en **repositorios remotos**, en el propio **POM** u otro relacionado o en **settings.xml**



Definición de Dependencias

- Especificación de una dependencia (biblioteca) en el pom.xml

```
<dependencies>
        ...
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.2</version>
            <scope>runtime</scope>
        </dependency>
    </dependencies>
    ...
</project>
```

Dependencias entre proyectos

- Las dependencias entre proyectos se manejan en los pom.xml de Maven especificando las coordenadas:

```
<dependencies>

 <dependency>
   <groupId>com.demo.maven</groupId>
   <artifactId>arq-core</artifactId>
   <version>1.0.0</version>
 </dependency>

</dependencies>
```

Dependencias Transitivas

- Tipos de dependencias en Maven
  - **Directas:** definidas en su archivo pom.xml, sección `<dependencies/>`
  - **Transitivas:** dependen de sus dependencias directas

En un árbol de dependencias como *A* → *B* → *C* , **C es una dependencia transitiva para A** (*B* tiene alcance *compilar* dentro de *A*)

10

Optimización de tareas

Despliegue de proyectos

- Agregando el proyecto al servidor
  - Para desplegar los proyectos se crea un Tomcat y se agrega al proyecto

Despliegue de proyectos

- **deployment assembly**
  - Si se desplegara el war del empaquetado en un Tomcat externo, con lo realizado es suficiente. Bastaría con copiar dicho war en la carpeta webapps.
  - Sin embargo, para poder desplegar en un Tomcat embebido en Eclipse, se debe asegurar que los proyectos dependientes y las dependencias de Maven estén en el "Deployment Assembly"

Despliegue de proyectos

- **targeted runtime**
  - Otra configuración que se debe tener en cuenta para el funcionamiento con un Tomcat embebido, es que el proyecto debe tener configurado Tomcat como Targeted Runtime.
  - Esto permite que se copien todas las dependencias, tanto de otros proyectos como externas, al directorio temporal que utiliza internamente Eclipse para desplegar

Otras utilidades

- dependency:sources
  - Obtiene el código fuente de las dependencias y lo coloca en el repositorio local. Eclipse puede leerlo.
- dependency:copy-dependencies
  - Copia todas las dependencias, incluyendo las transitivas, al directorio target/dependency. Es útil para herramientas que necesitan las dependencias, y no se integran con Maven.
- source:jar
  - Genera un jar con el código fuente del proyecto, y lo coloca en el directorio target.
- javadoc:jar
  - Genera un jar con el javadoc (apidocs) del proyecto, y lo coloca en el directorio target.

11

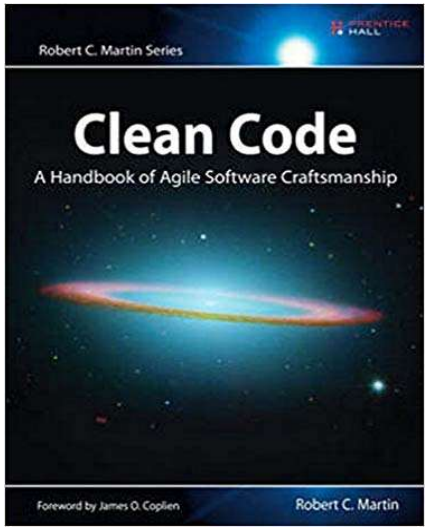
Mejora de la calidad del código

Clean Code

- Guía de **buenas prácticas** a la hora de escribir código:
  - Buenas prácticas de código limpio, qué hay que hacer para obtener código limpio.
  - Ejemplos de situaciones reales en las que aplicar lo explicado en la teoría.
  - Síntomas y heurísticas para detectar código no limpio, donde se repite gran parte del contenido de la teoría.

Clean Code: A Handbook of Agile Software Craftsmanship

(Robert C. Martin)



Clean Code: principales prácticas

- **Nombres con sentido**
  - usar variables con nombres descriptivos para explicitar nuestra intención, aunque el código quede más largo
  - usar nomenclatura estándar (nombres de patrones, nombres de funciones existentes tipo clone o toString, nombres del dominio…)
  - Evitar codificaciones en los nombres
- **Funciones**
  - Nombres inequívocos, mejor largos pero que expresen lo que hacen
  - Demasiados argumentos (máximo tres)
  - Argumentos de salida
  - Argumentos booleanos son indicadores de que la función hace más de una cosa
  - Función muerta (nadie la usa)

Clean Code: principales prácticas (II)

- **Comentarios**
  - Comentarios legales, de derechos de autor
  - Comentarios informativos de lo que devuelve una función, aunque también se pueden eliminar si en el nombre de la función especificamos lo que se devuelve
  - Explicación de la intención, decisión tomada o advertencia
  - Cuando utilizamos librerías de terceros, que no podemos modificar los nombres de funciones
  - Comentarios TODO, aunque no deben ser excusa para dejar código incorrecto
  - Comentarios en API públicas (Javadoc)

Clean Code: principales prácticas (III)

- **Objetos y estructuras de datos**
  - Clase (esconde su implementación interna) vs estructura de datos (los expone tal cual)
  - Ley de Demeter
- **Refinamiento sucesivo**
  - El refactor consiste en hacer cambios pequeños poco a poco y que en cada paso pequeño pasen todos los tests.

…

Clean Code: principales prácticas (IV)

- **Tests**
  - Hacer pruebas unitarias
  - Pruebas suficientes, se debe probar todo lo que pueda fallar
  - Usar herramientas de cobertura
  - Probar condiciones de límite
  - No ignorar pruebas triviales, sobretodo por su labor de documentación
  - Usar pruebas ignoradas (@Ignore) para preguntar sobre ambigüedades
  - Las pruebas deben ser rápidas

Clean Code: principales prácticas (V)

- **Procesar errores**
  - No usar códigos de error: confunden el flujo de ejecución y obligan al invocador a procesarlos inmediatamente.
  - En los errores incluir información que nos dé contexto de dónde se ha producido el fallo.
  - Al usar APIs de terceros siempre envolver excepciones (patrón Facade).
  - Crear clases para los casos especiales en lugar de dejar al código cliente procesar el caso excepcional (patrón caso especial, Fowler).
  - En general no es recomendable devolver null, en su lugar es mejor devolver una excepción o un objeto de caso especial

Clean Code: principales prácticas (VI)

- **Clases**
  - Orden dentro de la clase: Contantes estáticas, variables estáticas, variables de instancia y funciones (primero lo público y después lo privado).
  - El tamaño debe ser reducido, debe tener una única responsabilidad, la que indica su nombre.
  - **Single Responsibility** Principle, una clase debe tener un único motivo para cambiar.

Clean Code: principales prácticas (VII)

- **Clases**
  - **Cohesión** = Queremos clases cohesionadas. Cuando se reduce el tamaño de las funciones se aumenta el tamaño de variables de instancia (para no pasarlas como parámetro a las subfunciones) y se pierde cohesión. En ese caso lo mejor es dividir en subclases.
  - **Open/Closed** Principle = las clases deben estar abiertas a extensión y cerradas a modificación. Los cambios mejor que se hagan extendiendo o introduciendo nuevas clases, no modificando las existentes.
  - **Dependency Inversión** Principle = las clases dependen de abstracciones, no de detalles concretos –> Imprescindible para unit testing.

Test Drive Development (TDD)

El **desarrollo dirigido por pruebas** (TDD) es una práctica de ingeniería de software que involucra otras dos prácticas:

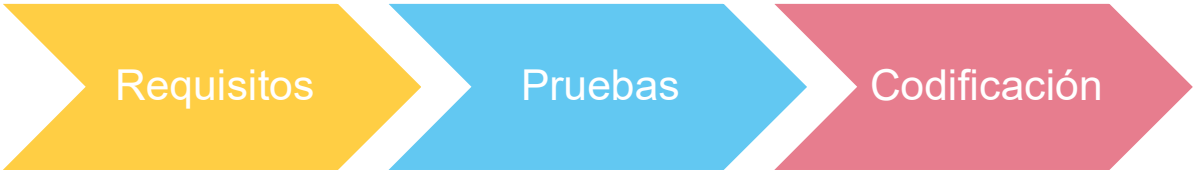
- Escribir las pruebas primero (**Test First Development**)
- Refactorización (**Refactoring**)

Objetivos:

- Minimizar el número de bugs.
- Implementar las funcionalidades justas que el cliente necesita.
- Producir software modular, altamente reutilizable y preparado para el cambio.

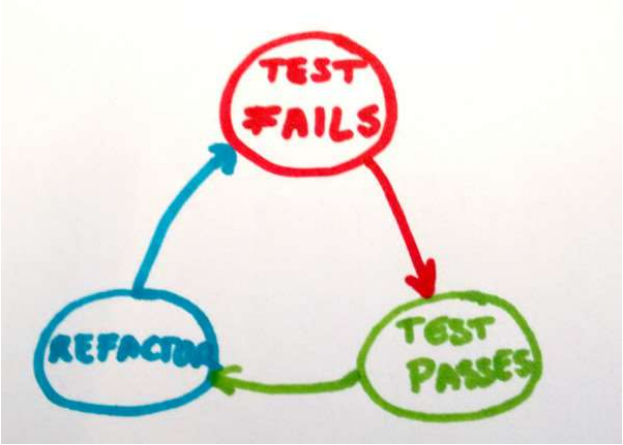
Flujo de trabajo tradicional

Flujo de trabajo con TDD



TDD: red-green-refactor

1. **ROJO:** escribir un test que falle
2. **VERDE:** escribir el mínimo código fuente necesario para hacer pasar el test
3. **AZUL:** refactorizar el código, limpiar, evitar redundancia de código, etc.



Beneficios de TDD

- Código probado desde el principio
- Todo el código bajo el control de las pruebas/tests
- No es necesario depurar código complejo
- Código de gran calidad
- Alta confianza en el código desarrollado

12

Generación de la documentación

Integración de Maven con Javadocs

- Generate Javadocs as part of Project reports

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
          ...
        </configuration>
      </plugin>
    </plugins>
    ...
  </reporting>
  ...
</project>
```

Generate Standalone Javadocs

- Add the Javadoc Plugin in the <build> section of your pom (if no configuration defined, the plugin uses default values):

```
<project>
 ...
 <build>
   <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-javadoc-plugin</artifactId>
       <version>3.2.0</version>
       <configuration>
         ...
       </configuration>
     </plugin>
   </plugins>
   ...
 </build>
 ...
</project>
```

Generate Standalone Javadocs

- And execute any of the following commands:

```
1. mvn javadoc:javadoc
2. mvn javadoc:jar
3. mvn javadoc:aggregate
4. mvn javadoc:aggregate-jar
5. mvn javadoc:test-javadoc
6. mvn javadoc:test-jar
7. mvn javadoc:test-aggregate
8. mvn javadoc:test-aggregate-jar
```

Javadoc Configuration

- The Javadoc Plugin supports a large number of configuration parameters.
- Each configuration parameter turns into a tag name.

1. `mvn site`
2. `mvn javadoc:javadoc`